The following is a complete listing of all claims in the application, with an indication of the status of each:

Listing of claims:

1. (currently amended) An automated method for converting tables of data in a source database to components of a target application, the method comprising the steps:

obtaining by a computer identification of a) specified tables in said source database containing data usable in said target application, b) a target location for said target application and c) an application server being used for development of said target application at said target location

reading by a computer definitions of said specified tables from said source database; and

generating by a computer from said database definitions a plurality of source code files, said source code files being generated directly in a language of said target application, each said specified table being referenced consistently across said plurality of source code files, said plurality of source code files including object classes and deployment descriptors, said specified tables being made accessible to a remote client by said target application, said target application being developed using said plurality of source code files.

2. (previously presented) The method of claim 1 further comprising the steps of:

providing a user interface to permit interaction with the computer application program;

including in said obtaining step identification of d) an Enterprise JAVA Bean Java Archive (EJB Jar) file;

generating said plurality of source code files in said identified target location, said identified target location being a user-specified directory path, said application server supporting development of Enterprise JAVA Beans (EJBs) and supporting a Java Naming and Directory Interface (JNDI) to assist with said consistent table references, and said plurality of source code files comprising

a Data JAVA file for each specified table, a Home Interface JAVA file for each specified table, a Remote Interface JAVA file for each specified table, a Bean JAVA file for each specified table, a Primary Key JAVA file for each specified table, a Persistent JAVA file for each specified table, an Enterprise JAVA Bean Deployment Descriptor XML file, an Enterprise JAVA Bean Jar batch command file, a Vendor-Specific Deployment XML file, and a Vendor-Specific Build batch command file.

3. (previously presented) The method of claim 2, wherein providing a user interface further comprises the steps of:

querying the database to get names of all tables related to the database, from which each of said specified tables are selected for encapsulation with an EJB; and

querying the database to acquire information about fields within each of said specified tables.

4. (previously presented) The method of claim 2, wherein generating the Data JAVA file further comprises the steps of:

generating a file with its filename equal to a table name followed by the word "Data" and file extension "java";

writing header comments indicating that a JAVA class encapsulates one row of the specified table;

writing a JAVA package definition consistent with the user-specified directory path;

writing a first JAVA statement to import a java.io.Serializable class;

writing a JAVA class definition with a class name equal to its filename;

writing a second JAVA statement indicating that the class implements the Serializable interface;

writing a corresponding attribute definition for each one of a plurality of fields in the specified table;

ensuring that a JAVA data type of each one of a plurality of attributes is compatible with a database data type of its corresponding field;

ensuring that a name of each one of the plurality of attributes is identical to the name of its corresponding field;

writing a first JAVA public method to get a value of each one of the plurality of attributes;

writing a second JAVA public method to set the value of each one of the plurality of attributes; and

writing a third JAVA public method named "toString" to return a string representing a Data class.

5. (previously presented) The method of claim 2, wherein generating the Home Interface JAVA file further comprises the steps of:

generating a file with filename equal to a table name followed by the word "Home" and file extension "java";

writing header comments indicating that a JAVA class is the Home Interface for the Bean encapsulating the specified table;

writing a JAVA package definition consistent with the user-specified directory path;

writing a first JAVA statement to import the following classes: javax.ejb.EJBHome; javax.ejb.FinderException; javax.ejb.CreateException; java.rmi.RemoteException; java.util.Collection;

writing a JAVA interface definition with an interface name equal to the filename;

writing a second JAVA statement indicating that the class extends the EJBHome interface;

writing a first JAVA method signature named "create" which takes as parameters the attributes of Data class and returns an object of Remote class;

writing a second JAVA method signature named "create" which takes as a parameter an object of Data class and returns an object of Remote class; and

writing a third JAVA method signature named "findByPrimaryKey" which takes as a parameter an object of PrimaryKey class and returns an object of Remote class wherein writing the JAVA method signature named "findByPrimaryKey" has one primary key and the computer application program makes the parameter its corresponding attribute of Data class.

6. (previously presented) The method of claim 2, wherein generating the Remote Interface JAVA file further comprises the steps of:

generating a file with its filename equal to the table name followed by the word "Remote" and file extension "java";

writing header comments indicating that this JAVA class is the Remote interface for the Bean encapsulating the specified table;

writing a JAVA package definition consistent with the user-specified directory path;

writing a first JAVA statement to import the following classes: javax.ejb.EJBobject; java.rmi.RemoteException;

writing a JAVA interface definition with its interface name equal to the filename;

writing a second JAVA statement indicating that the class extends the EJBObject interface;

writing a first JAVA public method signature to get a value of each attribute and wherein the signature indicates that the method may throw a Remote Exception; and

writing a second JAVA public method to set the value of each attribute signature and wherein the signature shall indicate that the method may throw a Remote Exception.

7. (previously presented) The method of claim 2, wherein generating the Bean JAVA file further comprises the steps of:

generating a file with its filename equal to its table name followed by the word "Bean" and file extension "java";

writing header comments indicating that the JAVA class is an EJB which encapsulates one row of the specified table;

writing a JAVA package definition consistent with the user-specified directory path;

writing a first JAVA statement to import the following classes: java.rmi.RemoteException; all classes in the javax.ejb package; all classes in the java.util package;

writing a JAVA class definition with its class name equal to the filename;

writing a second JAVA statement indicating that the class implements the EntityBean interface;

writing an attribute definition for one attribute named "theContext" of type "EntityContext";

writing, for each field in the specified table, its corresponding attribute definition;

ensuring that the JAVA data type of each attribute is compatible with the database data type of its corresponding field;

ensuring that each attribute is identical to the lower-case name of its corresponding field;

writing a first JAVA public method to get the value of each attribute;

writing a second JAVA public method to set the value of each attribute;

writing a third JAVA public method named "setAll" which has an object of Data class as parameter and can throw a Remote Exception and wherein each attribute of Bean class is set to the value of its corresponding Data attribute;

writing a fourth JAVA public method named "getAll" which returns an object of Data class and can throw a Remote Exception;

writing a fifth JAVA public method named "ejbCreate" with attributes of Data class as parameters and a signature of this method says it returns an object of type "String" but the body of the method must return "null" and this method can throw a Create Exception and a Remote Exception;

writing a sixth JAVA public method named "ejbPostCreate" with attributes of Data class as parameters and the signature of this method says it returns an object of type "String" but the body of the method must return "null" and this method can throw a Create Exception and a Remote Exception;

writing a seventh JAVA public method named "ejbPostCreate" with an object of Data class as the parameter and the signature of this method says it returns an object of type "String" but the body of the method must return "null" and this method can throw a Create Exception and a Remote Exception;

writing an eighth JAVA public method named "ejbLoad" which can throw an EJB Exception or a Remote Exception and the body of this method may be empty;

writing a ninth JAVA public method named "ejbStore" which can throw an EJB exception or a Remote exception and the body of this method may be empty;

writing a tenth JAVA public method named "ejbActivate" which can throw an EJB Exception or a Remote Exception and the body of this method can be empty;

writing an eleventh JAVA public method named "ejbPassivate" which can throw an EJB Exception or a Remote Exception and the body of this method can be empty;

writing a twelfth JAVA public method named "ejbRemove" which can throw an EJB Exception or a Remote Exception and the body of this method can be empty;

writing a thirteenth JAVA public method named "setEntityContext" which can throw an EJB exception or a Remote Exception and the parameter for this method is an object of type EntityContext and is used to set theContext;

writing a fourteenth JAVA public method named "unsetEntityContext" which can throw an EJB Exception or a Remote Exception and this method sets theContext to "null"; and

writing a fifteenth JAVA public method named "toString" to return a string representing the Bean class.

8. (previously presented) The method of claim 2, wherein generating the Primary Key JAVA file further comprises the steps of:

generating a file with at least two primary key fields such that its filename is made equal to its table name followed by the word "PrimaryKey" and file extension "java";

writing header comments indicating that this JAVA class encapsulates the Primary Key of the specified table;

writing a JAVA package definition consistent with the user-specified directory path;

writing a first JAVA statement to import the java.io.Serializable class;

writing a JAVA class definition with the class name equal to its filename;

writing a second JAVA statement indicating that its class implements the Serializable interface;

writing a corresponding attribute definition for each primary key field;

ensuring that the JAVA data type of each attribute is compatible with the database data type of its corresponding field;

ensuring that each attribute is identical to the lower case name of its corresponding field;

writing a first JAVA public method to get the value of each attribute;

writing a second JAVA public method to set the value of each attribute;

writing a third JAVA public method named "equals" which returns a boolean value and the parameter to this method is an object of the Object class and returns "true" if the parameter is an instance of the Primary Key class and if the parameter's attributes have values equal to the values of the attributes of this object; and

writing a fourth JAVA public method named "hashcode" which returns an integer value and this method forms a String of the attribute values and returns a hash code of that String.

9. (previously presented) The method of claim 2, wherein generating the Persistent JAVA file further comprises the steps of:

generating a file with filename equal to the table name followed by the word "Persistent" and file extension "java";

writing header comments indicating that this JAVA class encapsulates one row of the specified table and this client side class can be passed to the server for execution;

writing a JAVA package definition consistent with the user specified directory path;

writing a first JAVA statement to import the following classes javax.naming.InitialContext; javax.naming.NamingException; javax.rmi.PortableRemoteObject; java.sql.Connection; java.io.Serializable; all classes of the utility package com.lmco.util;

writing a JAVA class definition with the class name equal to the filename;

writing a second JAVA statement indicating that the class extends its corresponding Data class;

writing a third JAVA statement indicating that the class implements the Serializable interface and the PersistentObject interface;

writing a first JAVA public method to construct an object of this class without any parameters;

writing a second JAVA public method to construct an object of this class with the attributes of the Data class as parameters;

writing a third JAVA public method to construct an object of this class with an object of the Data class as parameters;

writing a fourth JAVA public method named "create" which returns a boolean and the body of this method provides a JAVA try-and-catch block to

invoke the "create" method of the Home interface with the attributes of this class as parameters and if an exception occurs, the result is false or otherwise the result is true;

writing a fifth JAVA public method named "read" which returns a boolean and the body of this method provides a JAVA try-and-catch block to invoke the "findByPrimaryKey" method of the Home interface with the Primary Key class as parameter and if an exception occurs, the result is false or otherwise, the result is true and the attributes of this class are set to the values in the Remote interface returned by the "findByPrimaryKey" method;

writing a sixth JAVA public method named "update" which returns a boolean and the body of this method provides a JAVA try-and-catch block to invoke the "findByPrimaryKey" method of the Home interface, with the primary key class as parameter and if an exception occurs, the result is false or otherwise, the result is true and the attributes of the Remote interface are set to the values of the attributes in this class;

writing an seventh JAVA public method named "delete" which returns a boolean and the body of this method provides a JAVA try-and-catch block to invoke the "remove" method of the Home interface with the Primary Key class as parameter and if an exception occurs, the result is false or otherwise the result is true; and

writing a JAVA protected method named "getHome" to return an object of the Home class and the body of this method providing a JAVA try-and-catch block to use an object of type InitialContext to look up in the JNDI a name, which JNDI name is equal to the name of the specified table.

10. (previously presented) The method of claim 2, wherein generating the EJB Deployment Descriptor XML file further comprises the steps of:

generating a file with filename "ejb-jar.xml" and located in a folder named META-INF one directory deep within the user-specified target directory path;

writing an XML header statement for documents of type "ejb-jar";

writing a first tag <ejb> to begin the document;

writing a second tag <description> to begin a description;

writing the description of a jar file which includes the package name and the identified EJB jar filename;

writing a third tag </description> to end the description;

writing a fourth tag <enterprise-beans> to begin a list of EJB's;

writing a fifth tag <entity> to begin a definition of its corresponding entity EJB;

writing a sixth tag <description> to begin the description of the EJB;

writing a description of the EJB including the name of its corresponding specified table;

writing a seventh tag </description> to end the description;

writing an eighth tag <ejb-name> to begin the name of the EJB;

writing the name of its corresponding specified table;

writing a ninth tag </ejb-name> to end the name of the EJB;

writing a tenth tag <home> to begin a home of the EJB;

writing a fully-qualified-name of its corresponding Home class;

writing an eleventh tag </home> to end the home of the EJB;

writing a twelfth tag <remote> to begin a remote interface of the EJB;

writing a fully-qualified-name of its corresponding Remote class;

writing a thirteenth tag</remote> to end the remote of the EJB;

writing a fourteenth tag <ejb-class> to begin a bean class of the EJB;

writing the fully-qualified-name of its corresponding Bean class;

writing a fifteenth tag </ejb-class> to end the bean class of the EJB;

specifying container-maintained persistence with the statement <persistence-type>Container</persistence-type>;

writing a sixteenth tag <prim-key-class> to begin the primary key class of the EJB;

writing the fully-qualified-name of its corresponding PrimaryKey class;

writing a seventeenth tag </prim-key-class> to end the primary key class of the EJB;

specifying the EJB as not re-entrant by writing <reentrant>False</reentrant>;

beginning each field of the specified table with <cmp-field><field-name>;

writing each field of the specified table;

ending each field of the specified table with </field name></cmp-field>;

if the specified table has only one primary key field, writing the primary key field as <primkey-field> followed by the field name followed by </primkey-field>;

writing an eighteenth tag </entity> to end a description of its corresponding entity EJB for each specified table;

writing a nineteenth tag <assembly-descriptor> to begin the assembly descriptor;

writing a twentieth or more tags defining a default security role;

providing default permission to all methods of all EJB's;

specifying container-managed transactions for all methods of each EJB;

writing a twenty-first or greater tag <assembly-descriptor> to end assembly;

writing a twenty-second or greater tag </enterprise-beans> to end the list of EJB's; and

writing a twenty-third or greater tag </ejb> to end the document.

11. (previously presented) The method of claim 2, wherein generating the EJB Jar batch command file further comprises the steps of:

generating a file with filename set to the identified name of the Jar file where the file extension on MS Windows-based systems is "bat";

writing a command to compile each generated Java file;

writing a command to put the ejb-jar.xml file into one Jar file; and

for each specified table, writing a command to put all of generated JAVA classes into the Jar file.

12. (previously presented) The method of claim 2, wherein generating the Vendor-Specific Deployment XML file further comprises the steps of:

generating a file with filename indicating both the vendor and the identified name of the EJB Jar file;

generating one or more XML tags according to vendor specifications for a target Application Server and for which the details will vary with the Vendor;

generating one or more XML tags to specify a JNDI name for the EJB corresponding to each specified table and the JNDI name is the same as its specified table name; and

generating one or more XML tags to specify mapping of each EJB attribute to its field in the specified database table.

13. (previously presented) The method of claim 2, wherein generating the Vendor-Specific Build batch command file further comprises the steps of:

generating a file with its filename indicating both a vendor and the word "command" and wherein the file extension on MS Windows-based systems is "bat";

generating vendor-specific commands to import and deploy a Jar file according to the Vendor-Specific Deployment file.